# Temporal Difference

**Problem we are trying to solve:**

The environment is that rewards usually are not immediately observable. For example, in tic-tac-toe or others, we only know the reward(s) on the final move (terminal state). All other moves will have 0 immediate rewards.

**Overview :**

TD learning is an unsupervised technique to predict a variable's expected value in a sequence of states. Instead of trying to calculate total future reward, TD simply tries to predict the combination of immediate reward and *its own reward prediction at the next moment in time*. Then, when the next moment comes, bearing new information, the new prediction is compared against what it was expected to be. If they're different, the algorithm calculates how different they are, and uses this "temporal difference" to adjust the old prediction toward the new prediction. By always striving to bring these numbers closer together at every moment in time – matching expectations to reality – the entire chain of prediction gradually becomes more accurate.

**Concept :**

TD methods are used for estimating value functions and improving policies in RL. The core idea behind TD methods is to update value estimates based on the difference between the predicted and actual rewards observed during an agent's interaction with the environment.

At each time step t, the agent updates its value estimate for a state or state-action pair based on the observed reward and its own estimate of the expected future rewards.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

**The TD Error** is the difference between the ultimate correct reward (V*_t) and our current prediction (V_t).

$$\begin{aligned} E_t &= V_t^* - V_t \\ &= r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} - V_t \\ &= r_{t+1} + \gamma * \left( \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k+1} \right) - V_t \\ &= r_{t+1} + \gamma * \left( \sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k+1} \right) - V_t \\ &= r_{t+1} + \gamma * V_{t+1} - V_t \end{aligned}$$

And similar to other optimization methods, the current value will be updated by its value + learning_rate * error:

$$V_t \leftarrow V_t + \alpha * E_t = V_t + \alpha * (r_{t+1} + \gamma * V_{t+1} - V_t)$$

# Algorithms

## *SARSA*

SARSA (State-Action-Reward-State-Action) is a reinforcement learning algorithm that is used to learn a policy for an agent interacting with an environment. It is a type of on-policy algorithm, which means that it learns the value function and the policy based on the actions that are actually taken by the agent.

> ***SARSA technique is an On Policy and uses the action performed by the current policy to learn the Q-value.***

Here we already choose next action and then we Update current Q value

$$Q(s_t, a_t) = Q(a_t, a_t) + \alpha(r_t + \gamma(Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)))$$

In Code :

```
a_ = max_action(s_) if rand < (1-epsilon) else env.action_space.sample()
Q[s,a] = Q[s,a] + alpha*(reward - Q[s,a] + gamma*Q[s_,a_])
```

## *Q-learning*

Q-Learning is a basic form of Reinforcement Learning which uses Q-values (also called action values) to iteratively improve the behavior of the learning agent.

***Q-Learning technique is an Off Policy technique and uses the greedy approach to learn the Q-value.***

Here we update Q-value predicting best future reward

$$Q(S, A) = Q(S_t, A_t) + \alpha(r_t + \gamma(\max_a Q(S_{t+1}, a) - Q(S_t, A_t)))$$

In Code :

```
Q[s,a] = Q[s,a] + alpha*(reward - Q[s,a]
          + gamma*np.max([Q[s_,a_] for a_ in range(possible_action)]))
```

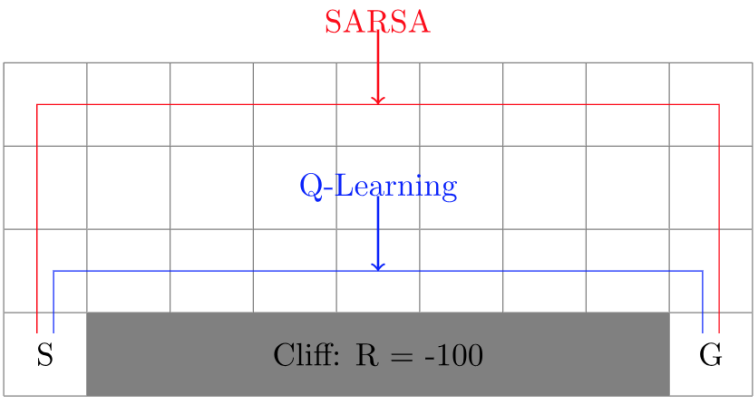
Q-learning differs from SARSA in its update rule by assuming the use of the optimal policy. The use of the function over the available actions makes the Q-learning algorithm an off-policy approach.This is because the policy we are updating differs in behavior from the policy we use to explore the world, which uses an exploration parameter to choose between the best-identified action and a random choice of action:

# The Difference

The difference is very subtle: For QL, which is the off-policy algorithm, when passing the reward from the next state (s_, a_) to the current state, it takes the **maximum possible reward of the new state (s_) and ignores whatever policy we are using**. For SARSA, which is on-policy, we still **follow the policy (e-greedy), compute the next state (a_)**, and pass the reward corresponding to that exact a_ back the previous step.

To reiterate, QL considers the best possible case if you get to the next state, while SARSA considers the reward if we follow the current policy at the next state. Hence, if our policy is greedy, SARSA and QL will be the same. But we are using e-greedy here, so there is a slight difference.

**QL directly learns the optimal policy while SARSA learns a “near” optimal policy.** QL is a more aggressive agent, while SARSA is more conservative. An example is walking near the cliff. QL will take the shortest path because it is optimal (with the risk of falling), while SARSA will take the longer, safer route (to avoid unexpected falling).




**Value update time example**: let's say we are in state (2,4) from there in Q-learning we will take max of all the future reward that we can get from all possible action → this will never be jumping of the cliff(3,4), this will only be updated mostly by (2,5),(2,3),(1,4) but in SARSA we take the action and the update the value. So, accordingly in this case if in the exploration phase if we jumped off the cliff this it will incur a large negative future reward to this state and hence will not be taken as optimal by SARSA algorithm.

**Reference**

***Take look at cliff edge example - to explicitly see the difference***

***Intro. to reinforcement learning - Temporal Difference***

***Es21btech11003***